Understanding ECDH Key Exchange in MeshCore

Elliptic Curve Diffie-Hellman (ECDH) is the cryptographic mechanism MeshCore uses to establish a shared secret between two nodes without that secret ever being transmitted over the radio. This page explains the underlying mathematics, describes how MeshCore uses ECDH in practice, and contrasts it with Meshtastic static PSK.

The Diffie-Hellman Principle

The Diffie-Hellman key agreement protocol solves a specific problem: how can two parties who have never met agree on a shared secret while communicating over a channel that an adversary can fully observe?

The classical analogy uses paint mixing. Alice and Bob both start with a public colour (yellow). Alice mixes in her secret colour (red) to get orange and sends orange to Bob. Bob mixes in his secret colour (blue) to get green and sends green to Alice. Alice adds her secret red to the green she received and gets a specific brownish mixture. Bob adds his secret blue to the orange he received and gets the same mixture. Both arrive at an identical colour without either secret colour ever being sent.

The mathematical version replaces paint with modular exponentiation in a finite group. The group structure makes forward computation easy but reversal computationally infeasible: the discrete logarithm problem.

The Elliptic Curve Variant

Classic Diffie-Hellman requires large key sizes (2048-4096 bits) for adequate security. Elliptic Curve DH achieves equivalent security with much shorter keys: a 256-bit ECC key provides roughly the same margin as a 3072-bit RSA key. This matters enormously for embedded LoRa nodes where flash, RAM, and CPU are scarce.

MeshCore node identities are Ed25519 keypairs (Curve25519-family). For the ECDH key exchange, the Ed25519 keys are transposed to their X25519/Curve25519 (Montgomery) form to compute the shared secret. Curve25519 was designed by Daniel J. Bernstein for high-performance constant-time

implementation on small processors. It avoids the implementation pitfalls such as timing side channels and weak curve parameters that have plagued other ECC curves. The public key is 32 bytes; the Ed25519 private key is 64 bytes (PRV_KEY_SIZE = 64). The 32-byte public key is stored in NVS and transmitted in advertisement packets.

In Curve25519 ECDH:

```
shared_secret = scalar_mult(my_private_key, their_public_key)
 = scalar_mult(their_private_key, my_public_key) // identical result
```

The scalar_mult function is a point multiplication on the elliptic curve. Computing it in the forward direction is efficient; reversing it to recover a private key from a public key and shared secret requires solving the elliptic curve discrete logarithm problem, for which no polynomial-time algorithm is known.

How MeshCore Uses ECDH in Practice

Step 1: Static Keypair Generation

At first boot, each MeshCore node generates an Ed25519 private key (64 bytes) from the platform hardware RNG and derives the corresponding 32-byte public key. Both are written to non-volatile storage and persist across reboots. These are static keypairs because they remain fixed for the lifetime of the firmware installation, as opposed to the ephemeral keypairs used per-session in TLS 1.3.

Step 2: Public Key Advertisement

The public key is included in the node periodic advertisement packet and propagated hop-by-hop across the mesh using MeshCore controlled-flood mechanism. Over successive advertisement cycles, nodes learn the public keys of peers whose adverts they receive and store them in their contact list. No prior direct contact is required before a secure direct message can be sent.

Step 3: Shared Secret Derivation on Demand

When node A wants to send an encrypted direct message to node B, it computes:

```
shared_secret_AB = Curve25519(A_private_key, B_public_key)
```

When node B receives the message, it computes:

```
shared_secret_AB = Curve25519(B_private_key, A_public_key)
```

Both operations produce the same 32-byte value. This shared secret is used as the AES-128 key (16 bytes of it) used to encrypt and decrypt the message body; the full 32-byte secret also keys the HMAC. MeshCore uses AES-128, not AES-256. The shared secret is cached in RAM after first derivation to avoid recomputing it on every subsequent message to the same peer.

Step 4: AES Encryption with the Derived Key

The derived AES-128 key encrypts the payload in ECB mode with zero-padding on the final block. MeshCore does not use CTR or GCM mode and there is no per-packet nonce; ECB encrypts each 16-byte block independently. A timestamp embedded in each message body helps vary the plaintext. Note: ECB leaks equality of identical 16-byte plaintext blocks, a known weakness. After encryption, a 2-byte truncated HMAC-SHA256 MAC (encrypt-then-MAC) is prepended to the ciphertext, and the result is placed in the packet payload and transmitted.

Comparison with Meshtastic Static PSK

Property	MeshCore ECDH (direct messages)	Meshtastic Static PSK
Key material transmitted over radio	Public keys only; shared secret never transmitted	PSK never transmitted but must be distributed out-of-band to all participants
Key uniqueness per pair	Each node pair has a unique shared secret	All nodes in channel share the same key
Compromise of one device	Exposes messages to and from that device only	Exposes all channel messages past and future

Property	MeshCore ECDH (direct messages)	Meshtastic Static PSK
Forward secrecy	None: keys are static (not ephemeral), so disclosure of a private key retroactively decrypts all recorded traffic for that node.	None: historical traffic decryptable if PSK obtained
Key rotation	Reflash firmware or clear NVS to generate new keypair	Change PSK on all channel members simultaneously
Setup complexity	Automatic: keys generated at boot and exchanged via mesh advertisements	Manual: PSK must be configured identically on all nodes
CPU cost per message	AES only after first exchange; ECDH result cached per peer	AES only
Effective against passive recording plus later key disclosure	No — keys are static, not ephemeral. If a node's private key is ever recovered, all previously recorded traffic to and from that node can be decrypted. MeshCore direct messages do NOT provide forward secrecy.	No: PSK disclosure retroactively decrypts all recorded traffic

There is no key-revocation mechanism. If a device is lost or stolen, other nodes continue to trust and encrypt to its public key until each is manually updated. Plan for manual contact-list cleanup after any device compromise. To rotate a node's own keys you must reflash firmware or clear NVS to generate a new keypair.

Practical Implications for Message Privacy

For direct messages, the ECDH model means two nodes can communicate privately without pre-arranging any shared secret. An eavesdropper who captures every radio packet including the advertisement floods carrying both public keys cannot reconstruct the shared secret and cannot decrypt the messages.

The key operational risk is device compromise. The private key stored in NVS flash is the single point of failure for a node message privacy. On platforms without hardware-enforced flash encryption, physical access to a device is equivalent to possessing the private key. Treat any captured or unaccounted-for device as a key compromise event and reflash it with new keys before returning it to service.

The ECDH model also provides an implicit mutual authentication property. Because ECDH produces the correct shared secret only when both private keys are used, impersonating node B to node A requires producing ciphertext derivable from B private key, which is infeasible without that key. An adversary who has only the public key cannot derive the shared secret and so cannot trivially spoof

a node identity. Two caveats apply: the on-wire MAC is only 2 bytes (truncated HMAC-SHA256), which gives roughly 1-in-65,536 forgery resistance per attempt and is weak against active forgery attempts; and this authentication is only as good as the public key you hold. MeshCore learns keys automatically from unsigned-name advertisement floods with no out-of-band verification, so an attacker who injects an advert with their own key under a chosen name can be added as a contact. Verify a contact's full public key out-of-band before trusting direct-message authentication.

Revision #4
Created 2026-05-03 06:20:54 UTC by Mesh America Admin
Updated 2026-06-09 14:38:25 UTC by Mesh America Admin